Hash maps

CS 261 Lab #9
Hash maps are **awesome**

With an **appropriate hash function** and **sufficient size**, they give us $O(1)$ inserts, removes, and searches!

And they’re **associative arrays**, letting us work with **key/value** pairs
Hash maps are **awesome** true story

With an **appropriate hash function** and **sufficient size**, they give us $O(1)$ inserts, removes, and searches!

And they’re **associative arrays**, letting us work with **key/value** pairs
We’ll use the **buckets and chaining** approach to build hash maps
We’ll use the **buckets and chaining** approach to build hash maps.
We’ll use the **buckets and chaining** approach to build hash maps.
To **insert** a key/value pair...
To *insert* a key/value pair...

1) **Compute the hash** for the key
To *insert* a key/value pair...

1) **Compute the hash** for the key

2) **Determine the bucket** in the hash map this key/value pair belongs in

\[(\text{hash} \mod \# \text{ of buckets})\]
To **insert** a key/value pair...

1) **Compute the hash** for the key

2) **Determine the bucket** in the hash map this key/value pair belongs in
   \[\text{hash mod \# of buckets}\]

3) If the bucket doesn’t have any items in it, **make the bucket point to our item**. If it does have other items, **add our item to the chain**.
To lookup a value...
To *lookup* a value...

1) **Compute the hash** for the key
To *lookup* a value...

1) **Compute the hash** for the key

2) **Determine the bucket** in the hash map this key/value pair belongs in

\[(\text{hash} \mod \# \text{ of buckets})\]
To lookup a value...

1) **Compute the hash** for the key

2) **Determine the bucket** in the hash map this key/value pair belongs in 
\[(hash \ mod \ # \ of \ buckets)\]

3) Examine each key/value pair in the bucket until you **find the matching key**, then return the key’s value.
Example: adding the key “hello” with a value of “world”
Example: adding the key “hello” with a value of “world”

Hash “hello” to a number (e.g., 581)
Example: adding the key “hello” with a value of “world”

Hash “hello” to a number (e.g., 581)

Mod that number by the number of buckets in our hash map (e.g., 581 % 5 = 1)
Example: adding the key “hello” with a value of “world”

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Hash “hello” to a number (e.g., 581)

Mod that number by the number of buckets in our hash map (e.g., 581 % 5 = 1)

Add the key “hello” with a value of “world” to bucket 1
Example: adding the key “hello” with a value of “world”

Hash “hello” to a number (e.g., 581)

Mod that number by the number of buckets in our hash map (e.g., 581 % 5 = 1)

Add the key “hello” with a value of “world” to bucket 1
In today’s lab you’ll implement the **insert** and **contains** methods for a hash map.
In today’s lab you’ll implement the **insert** and **contains** methods for a hash map.

Then you’ll use the hash map to **build** a spell-checker.
In today’s lab you’ll implement the `insert` and `contains` methods for a hash map.

Then you’ll use the hash map to build a spell-checker.

This is a `very simplified` version of the hash map you’ll need to implement in Assignment #6 (but with a better hashing function!)
Your program will need to read the *dictionary.txt* file

If you use the **Makefile** to compile this lab (via Xcode, Eclipse, or manually), place *dictionary.txt* in the same directory as the Makefile

**Visual Studio** users will need to place *dictionary.txt* in Visual Studio’s build directory (usually *yourProjectDirectory\Debug*)
Download the files from http://classes.engr.oregonstate.edu/eecs/spring2015/cs261-001/lab9.php

Implement the `insertMap`, `containsKey`, and `tableLoad` functions in `hashmap.c`

In `main.c`, **use the hash map** to check whether words the user enters are spelled correctly

Experiment with different values of `size` in `main.c` to see how it **impacts the hash map’s speed and table load**